Lambda Expressions in Java 8

Lambda expressions basically express instances of functional interfaces (An interface with single abstract method is called functional interface. An example is java.lang.Runnable). lambda expressions implement the only abstract function and therefore implement functional interfaces lambda expressions are added in Java 8 and provide below functionalities.

- Enable to treat functionality as a method argument, or code as data.
- A function that can be created without belonging to any class.
- A lambda expression can be passed around as if it was an object and executed on demand.

Without Lamda

```
interface Drawable{
    public void draw();
}
public class LambdaExpressionExample {
    public static void main(String[] args) {
       int width=10;

        //without lambda, Drawable implementation usi
ng anonymous class
        Drawable d=new Drawable(){
           public void draw(){System.out.println("Dra
wing "+width);}
```

```
        };
         d.draw();
    }
}
```

With Lambda

```
@FunctionalInterface  //It is optional
interface Drawable{
   public void draw();
}

public class LambdaExpressionExample2 {
    public static void main(String[] args) {
      int width=10;

       //with lambda
       Drawable d2=()->{
          System.out.println("Drawing "+width);
      };
              d2.draw();
    }
}
```

```
// Java program to demonstrate lambda expressions
// to implement a user defined functional interface.

// A sample functional interface (An interface with
// single abstract method
interface FuncInterface
{
	// An abstract function
	void abstractFun(int x);

	// A non-abstract (or default) function
	default void normalFun()
	{
	System.out.println("Hello");
	}
}
```

```
class Test
{
	public static void main(String args[])
	{
		// lambda expression to implement above
		// functional interface. This interface
		// by default implements abstractFun()
		FuncInterface fobj = (int x)->System.out.println(2*x);

		// This calls above lambda expression and prints 10.
		fobj.abstractFun(5);
	}
}
```

Syntax:

lambda operator -> body

where lambda operator can be:

- **Zero parameter:**
  () -> System.out.println("Zero parameter lambda");
- **One parameter:**–
  (p) -> System.out.println("One parameter: " + p);

It is not mandatory to use parentheses, if the type of that variable can be inferred from the context

- **Multiple parameters :**

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

```
// A Java program to demonstrate simple lambda expressions
import java.util.ArrayList;
class Test
{
    public static void main(String args[])
    {
        // Creating an ArrayList with elements
        // {1, 2, 3, 4}
        ArrayList<Integer> arrL = new ArrayList<Integer>();
        arrL.add(1);
        arrL.add(2);
        arrL.add(3);
        arrL.add(4);

        // Using lambda expression to print all elements
```

```
            // of arrL
            arrL.forEach(n -> System.out.println(n));

            // Using lambda expression to print even
elements
            // of arrL
            arrL.forEach(n -> { if (n%2 == 0)
System.out.println(n); });
        }
}

interface MyInterface {

    // abstract method
    String reverse(String n);
}

public class Main {

    public static void main( String[] args ) {

        // declare a reference to MyInterface
        // assign a lambda expression to the reference
```

```
        MyInterface ref = (str) -> {

            String result = "";
            for (int i = str.length()-1; i >= 0 ; i--)
            result += str.charAt(i);
            return result;
        };

        // call the method of the interface
        System.out.println("Lambda reversed = " +
ref.reverse("Lambda"));
    }

}
```